Videojuegos

Curso de Diseño y Programación

Nº 12 5,99 euros

*Creación de música con **Anvil Studio***

*Animando con **Blitz3D***

***Implementar** el cielo en un entorno*

MULTIPLAYER SETUP

PLAYER SETUP

12







Curso práctico de Videojuegos

AUTOR DE LA OBRA
Marcos Medina
DIRECCIÓN EDITORIAL
Eduardo Toribio
etoribio@iberprensa.com
COORDINACIÓN EDITORIAL
Eva-Margarita García
eva@iberprensa.com
DISEÑO Y MAQUETACIÓN
Antonio Gª Tomé
PRODUCCIÓN
Marisa Cogorro
SUSCRIPCIONES
Tel: 91 628 02 03
Fax: 91 628 09 35
suscripciones@iberprensa.com
FILMACIÓN: Fotpreim Duvial
IMPRESIÓN: Gráficas Don Bosco
DUPLICACIÓN CD.ROM: M.P.O.
DISTRIBUCIÓN
S.G.E.L.
Avda. Valdelaparra 29 (Pol. Ind.)
28108 Alcobendas (Madrid)
Tel.: 91 657 69 00

EDITA: Iberprensa
www.iberprensa.com


CONSEJERO
Carlos Peropadre

REDACCIÓN, PUBLICIDAD Y ADMINISTRACIÓN
C/ del Río Ter, 7 (Pol. Ind. "El Nogal")
28110 Algete (Madrid)
Tel.: 91 628 02 03
Fax: 91 628 09 35
(Añada 34 si llama desde fuera de España.)

DEPÓSITO LEGAL: M-35934-2002
ISBN: Coleccionable: 84 932417 2 5
Tomo 2: 84 932417 4 1
Obra Completa: 84 932417 5 X

NOTA IMPORTANTE:
Algunos programas incluidos en los CD de "Programación y Diseño de Videojuegos" son versiones completas, pero en otros casos se trata de versiones demo o trial, versiones de evaluación que Iberprensa quiere ofrecer a nuestros lectores. No se trata en ningún caso de las versiones comerciales de los programas, y las hemos incluido para dar al lector la oportunidad de conocer y probar esos programas y que así pueda decidir posteriormente si desea o no adquirir las versiones comerciales de cada uno.

Aprende divirtiéndote

Bienvenidos de nuevo a **Programación y Diseño de Videojuegos**, la primera obra coleccionable cuyo objetivo es formar al alumno en las principales técnicas relacionadas en el desarrollo completo de un videojuego.

A lo largo de la obra el lector está aprendiendo programación a nivel general y a nivel específico con ciertas herramientas y lenguajes, aprendiendo a trabajar con aplicaciones de retoque de imagen y también de diseño 3D y animación. Estamos descubriendo las aplicaciones profesionales más importantes de audio y conociendo la historia de lo que se denomina "la industria del videojuego", los últimos 20 años, los juegos que marcaron un avance, sus creadores y en general la evolución del videojuego.

Pero además, esta obra tiene un segundo objetivo, desarrollar y potenciar la creatividad del lector, nosotros a lo largo de las diferentes entregas pondremos las bases y tú pondrás tu ingenio, tu creatividad y tu capacidad de mejorar.

Nos encontramos a mitad de camino del viaje de 20 semanas que os proponemos, viaje articulado en 400 páginas y 20 CD-ROMs cuya finalidad es proporcionar las bases mínimas para después cada uno continuar su camino.

Recuerda que para alcanzar el éxito necesitas cumplir tres condiciones: que te gusten los juegos, poseer cierta dosis de creatividad y finalmente capacidad de estudio.

Una la cumples seguro.

sumario



PARA ENCUADERNAR LA OBRA:

- *Para encuadernar los dos volúmenes que componen la obra "Programación y Diseño de Videojuegos" se pondrán a la venta las tapas 1 y 2.*
- ***Tapas del volumen 2 ya a la venta.***
- ***Los suscriptores recibirán las tapas en su domicilio sin cargo alguno como obsequio de Iberprensa.***

SERVICIO TÉCNICO:
Para consultas, dudas técnicas y reclamaciones Iberprensa ofrece la siguiente dirección de correo electrónico: games@iberprensa.com

PETICIÓN DE NÚMEROS ATRASADOS:
El envío de números sueltos o atrasados se realizará contra reembolso del precio de venta al público más el coste de los gastos de envío. Pueden ser solicitados en el teléfono de atención al cliente 91 628 02 03

Controlando al protagonista (II). Acciones

En el número anterior estudiamos los procedimientos para mover la bionave y cómo trabajar con la cámara.

En esta entrega completaremos la implementación del control del protagonista principal de nuestro juego. Fabricaremos las rutinas que harán posible las acciones de disparo, activación del camuflaje y del escudo.

```
Type Disp
  Field sprite
  Field alfa
  Field y_disparo#
  Field alcance
  Field elevacion#=20.0
  Field luz
  Field tipo_disparo
  Field retardo%
End Type
```

1

Estructura general utilizada para todos los disparos.

(■) EL DISPARO

Todas las rutinas de control de acciones están situadas en el módulo "Jugador_principal.bb". En este módulo, además, implementaremos la creación de los disparos y su evolución en el juego.

(■) DEFINICIÓN DEL TIPO DISPARO

Antes de continuar, es obligatorio crear previamente en el módulo "Definiciones.bb" la estructura de datos que definirá los disparos, así como todas las variables y estructuras necesarias para dar cobertura a su funcionamiento y evolución como son: las manipuladoras de entidades, controladoras de munición, explosiones, etc. (Fig. 1) (Ver Código 1).

(■) CONTROL DEL DISPARO

La acción de disparar se produce cuando el jugador pulsa el botón izquierdo del ratón o la tecla "Ctrl" derecha. Además, es posible cambiar de arma y para ello utilizamos el botón central del ratón o la tecla "Espacio" (por si el ratón del usuario carece de tres botones).

También contemplaremos la opción de intercambiar los botones derecho e izquierdo para jugadores zurdos (Ver Fig. 2). Es de fácil programación y una opción siempre de agradecer.

```
; Cambiar botones para los zurdos
; con F10, intercambiando las
; variables "boton1" y "boton2"
If KeyDown(68) And zurdos=0
    zurdos=1
    boton1=2: boton2=1
    Delay 300
EndIf
If KeyDown(68) And zurdos=1
    zurdos=0
    boton1=1: boton2=2
    Delay 300
EndIf
```

2

Código necesario para intercambiar los botones de control del ratón para zurdos.

Código 1. Definición de los disparos

```
Type disp
  Field sprite                        ; Campo para la entidad del sprite
  Field alfa                          ; Campo para controlar el canal alfa (transparencia)
  Field y_disparo#                    ; Altura del disparo
  Field alcance,elevacion#=20.0       ; Alcance del disparo y elevación máxima
  Field luz                           ; Manipulador de la luz creada en la explosión
  Field tipo_disparo                  ; Tipo de disparo según arma elegida
  Field retardo%                      ; Tiempo de retardo para explosionar las bombas de retardo
End Type
; Coordenadas generales para posicionar la creación de disparos en cada cañón de la bionave.
Global x_disparoA#=12                 ; Para vista en tercera persona
Global x_disparoB#=30                 ; Para vista subjetiva
Type aguje                            ; Estructura para los agujeros en la pared dejado por los disparos
  Field alfa#,sprite
End Type
Type explo                            ;Estructura para las explosiones
  Field alfa#,sprite,luz
End Type
Global disparo1,disparo2,disparo3,disparo4,explosion1,sprite_agujero
Global disparo1B,disparo2B,disparo3B,disparo4B
Global carga=0, tiempo_carga=10, frecuencia_disparo=8, cambio=0, tiempo_cambio=10
```

Código 2. Condicionales para crear disparos

```
If (MouseDown(boton1) Or KeyDown(157) And carga=0
  Select Tipo_armamento
    Case 1
      Crear_disparo.disp(pivote_bionave,tipo_armamento,disparo1)
      municion=municion-1
    Case 2
      Crear_disparo.disp(pivote_bionave,tipo_armamento,disparo2)
      municion2=municion2-1
    Case 3
      Crear_disparo.disp(pivote_bionave,tipo_armamento,disparo3)
      municion3=municion3-1
    Case 4
      Crear_disparo.disp(pivote_bionave,tipo_armamento,disparo4)
      municion4=municion4-1
    End Select
  carga=1
EndIf
```

Situándonos en la función "Control_jugador_principal()" añadimos las sentencias condicionales necesarias para la creación de los disparos (Ver código 2).

La variable "carga" es utilizada como activador para el control de la recarga del arma correspondiente. Si no se utilizase, cada vez que el jugador pulsara el botón no se controlaría la creación de disparos en todas las situaciones. Así que, mientras la recarga no se haya completado, no será posible otro disparo. Hacemos un inciso antes de continuar para explicar el control de recarga. Este control lo implementamos en el módulo de control del juego "Fjuego.bb" en la función "Actualizar_juego()". También debemos utilizar un contador que controle el tiempo de carga "Tiempo_carga", el cual se inicializará cuando llegue a 0 con un valor mayor o menor controlado por la variable "Frecuencia_disparo" y el poder del arma seleccionada ("Tipo_armamento") (Ver código 3).

De vuelta al módulo de control del jugador, una vez pulsado el botón de disparo entramos en una estructura "Select .. Case" para crear uno u otro disparo según el tipo de arma seleccionada a través de la variable "Tipo_armamento".

La creación de cada disparo la realizaremos a través de la función "Crear_disparo":

```
...
  Crear_disparo.Disp
(pivote_bionave, tipo_armamento,
disparo1)
  municion=municion-1
...
```

Realmente, esta función crea un nuevo objeto "Tipo" para la colección de la estructura "Disp".

La expresión "Crear_diparo.Disp (...)" es como si escribiéramos *Nuevo elemento. Tipo disparo = New Tipo disparo*; es decir *Disparo.Disp = New Disp*.

Es una forma de utilizar funciones para la creación de objetos "Type" individualmente cada vez que se le llaman. Además, debemos pasarle los parámetros necesarios para completar la estructura. De este modo, para la entidad en sí tenemos "disparo1" (entidad) y "pivote_bionave" (entidad madre) y para clasificarla "tipo_armamento" para saber qué clase de disparo se ha creado. Es importante resaltar que utilizamos un pivote creado a partir de la bionave para que los disparos se creen a partir de ella. El procedimiento de creación del pivote lo implementamos en el momento de cargar el fichero y crear el modelo en la función "Crear_modelos()" del módulo "Funcpantaudio.bb".

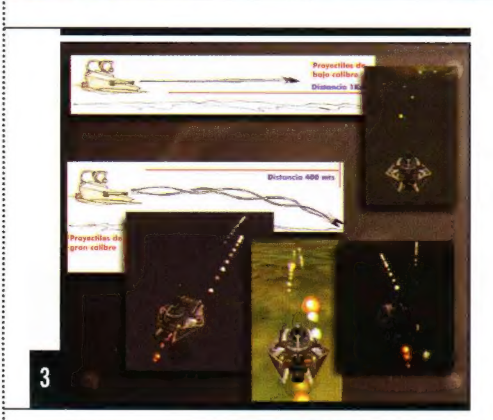
```
Function crear_modelos()
  Bionave = LoadMesh ("c:\zone
of fighters\modelos\bionave.3ds")
  pivote_bionave = CreatePivot
(bionave)
...
```

(■) CREANDO LOS DISPAROS. FUNCIÓN "CREAR_DISPARO()"

La función "Crear_disparo()" será la encargada de producir un nuevo objeto "Type" para la colección de la estructura "Disp". Para ello, establecemos una estructura "Select .. Case" para distinguir el tipo de disparo según el armamento elegido en el argumento de función "tipo_disp".

Código 3. Control del tiempo de carga

```
If Tiempo_carga=0 Then
  carga=0
  Select Tipo_armamento
    Case 1  Tiempo_carga=frecuencia_disparo              ; Tiempo de carga estándar
    Case 2  Tiempo_carga=frecuencia_disparo+4            ; Tiempo de carga estándar + 4
    Case 3  Tiempo_carga=frecuencia_disparo*2            ; El doble del tiempo estándar
    Default Tiempo_carga=(frecuencia_disparo*2)+5        ; El doble + 5
  End Select
EndIf
Tiempo_carga=Tiempo_carga-1                              ; Decrementamos el tiempo de carga
```



3

Diseño y resultado de la munición de bajo calibre (Tipo 1, estándar) y de los misiles (Tipo 2).

En las figuras 3 y 4 se muestran los diferentes tipos de armamento utilizado en el juego.

De los cuatro tipos disponibles explicaremos sólo los proyectiles de medio calibre (estándar), ya que para los demás, el procedimiento básico es el mismo (Ver código 4).

En el caso primero ("tipo_disp=1") creamos el nuevo disparo ("disparo.disp=New disp") y a continuación asignamos todos los valores a los campos de la estructura. En la asignación "disparo\sprite = CopyEntity (entidad_disparo, pivote_bionave)" hacemos una copia de la entidad original utilizada como disparo, en este caso un "sprite" y le asignamos como "entidad madre" el pivote de la bionave. Para lograr que los disparos salgan alternativamente de los dos cañones, utilizamos el comando "TranslateEntity" aprovechando la existencia de un pivote; es decir, con esta instrucción desplazamos la coordenada X de la posición de creación del disparo con referencia al centro de la bionave "TranslateEntity disparo\sprite,x_disparoB,1,-80".

Para lograr desplazar el punto hacia cada cañón cada vez que se cree un disparo utilizamos "x_disparoB = - x_disparoB" (desplazamiento de X en 30 puntos hacia la izq. una vez y 30 hacia la dcha. la siguiente vez) (Fig. 5).

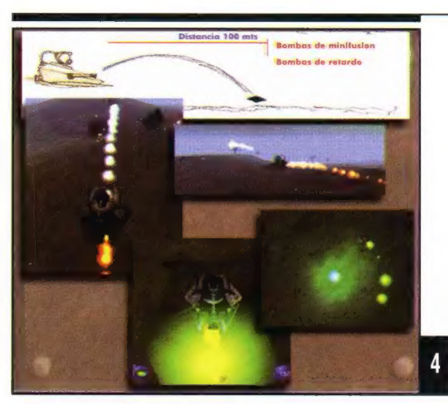
Una vez creado el disparo es importante liberarlo de la entidad madre, en este caso del pivote o centro de la bionave, para que se mueva autónomamente. Para ello, utilizamos la instrucción "EntityParent" con argumento 0 para asignar una entidad madre nula:

```
"EntityParent disparo\sprite, 0"
```

Una vez creado el disparo debemos devolver con la instrucción "Return" el nuevo objeto de la estructura creado ("Disparo").

Código 4. Creando los disparos

```
Function Crear_disparo.disp(pivote_bionave,tipo_disp,entidad_disparo)
  Select tipo_disp
    Case 1  ;Proyectiles medio calibre
      If municion<1 Return
      disparo.disp=New disp
      disparo\alfa=-90
      disparo\sprite=CopyEntity(entidad_disparo,pivote_bionave)
      disparo\alcance=300*tipo_disp  ;300 mts. calibre bajo, 600 mts
      gran calibre
      disparo\elevacion=0
      If tipo_camara=1       ; Vista subjetiva
        TranslateEntity disparo\sprite,x_disparoB,1,-80
        x_disparoB=-x_disparoB
        Else                 ; Vista en tercera persona
        x_disparoA=-x_disparoA
        TranslateEntity disparo\sprite,x_disparoA,-.5,-10
      EndIf
      EntityParent disparo\sprite,0
    Case 2 .........
    Case 3 .........
    Case 4 .........
    Return disparo
End Function
```



4

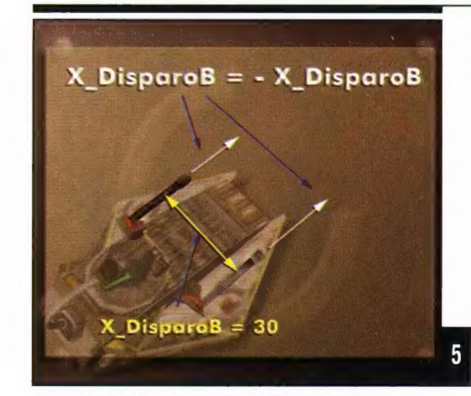
Diseño y resultado de la bomba de minifusión (Tipo 3) y de la bomba de retardo (Tipo 4).

(■) ACTUALIZANDO LOS DISPAROS. FUNCIÓN "ACTUALIZAR_DISPARO()"

Creado el disparo, es necesario actualizar su existencia y evolución en el juego. Para ello, utilizaremos la función "Actualizar_disparo()". En ella, básicamente, desplazamos la entidad del disparo hacia su destino y controlamos su colisión con el entorno (terreno, decorado, seres, jugadores y otros disparos).

Para actualizar todos los disparos creados durante el juego es necesario recorrer la colección de datos de la estructura del tipo "Disp" y llamar a la función para cada elemento "disparo".

Este control debemos aplicarlo por cada ciclo del control de partida; es decir, en la función "Actualizar_juego()" del módulo "Fjuego.bb":



5

Esquema gráfico del funcionamiento de la función "TranslateEntity" para el disparo alternativo en ambos cañones.


6

Algunas imágenes del desarrollo de algunas explosiones.

```
For disparo.disp = Each Disp
    Actualizar_disparo(disparo)
Next
```

Llamamos a la función pasándole como parámetro el objeto "type" del disparo. De vuelta al módulo "Jugador_principal()", analizaremos la función "Actualizar_disparo()" anterior:

(Ver función: Function Actualizar_disparo (disparo.disp).

En un principio averiguamos si ha llegado al límite de su alcance para crear una explosión y eliminarla, a través del campo "alcance". Seguidamente, realizaremos todos los controles de colisión con el comando "EntityCollided". El sistema de control de colisiones es muy extenso y complejo, así que realizaremos un estudio más detallado de su funcionamiento en el próximo número.

Después de controlar las colisiones, debemos desplazar el disparo y actualizar la variación de su aspecto durante el movimiento. Utilizamos una estructura "Select .. Case" para trabajar con cada tipo de disparo. Tomemos como ejemplo el tipo estándar ("disparo\tipo_disparo=1"). En primer lugar, modificamos el tamaño del "sprite" por medio de una función seno "Sin" según su actual transparencia ("disparo\alfa") Rotamos y desplazamos el "sprite" y decrementamos su alcance.


7

Imágenes del sistema de escudo en funcionamiento.

(■) EXPLOSIONES. FUNCIONES "CREAR_EXPLOSION()" Y "ACTUALIZAR_EXPLOSION"

Siempre que un disparo llega a su destino, bien por un impacto o bien por llegar al límite de su alcance, se produce una explosión. Como siempre, el aspecto de la explosión dependerá del tipo de disparo que la genere. Llamamos a la función "Crear_explosion()" pasándole como parámetro el objeto "type" "disparo" (*Crear_explosion(disparo)*) para poder utilizar el mismo sprite como entidad: *Function Crear_explosion.explo(disparo.disp*. Su funcionamiento es simple. En primer lugar creamos el nuevo objeto de tipo "explo" y asignamos un valor de transparencia. Seguidamente, según el tipo de disparo, damos los valores a los campos de la estructura "explo": creamos la entidad, definimos su presencia en el mapa de colisiones y posicionamos una luz y un emisor de partículas en sus coordenadas actuales. (Ver función *Function Crear_explosion. explo(disparo.disp)*).

Las explosiones debemos actualizarlas al igual que los disparos; es decir, en la función "Actualizar_juego()" del módulo "Fjuego.bb", llamando a la función de actualización (Ver función *Function actualizar_explosion (explosion.explo)*).

Básicamente, su funcionamiento consiste en aumentar y disminuir el tamaño del sprite de la explosión según el valor del campo "alfa", el cual, aumenta de 10 en 10 ("explosion\alfa=explosion\alfa+10").

(■) ESCUDO Y CAMUFLAJE

Mediante la tecla de control izquierda activamos el escudo de la bionave y con "Alt" el camuflaje. Ambas características están regidas por una variable que regula su disponibilidad de activación. Además, por medio de "Switches" ("escudo_bionave" y "camuflaje_bionave") avisamos de su activación y también evitamos, por ejemplo, varios escudos a la vez. Estas variables nos servirán después en la función "actualizar_jugador_principal()" para llamar a sus correspondientes funciones:

```
If escudo_bionave=
1 escudo_bionave().
```

(■) ACTIVACIÓN DEL ESCUDO

Al activar el escudo, en la función "control_jugador_principal()", creamos una esfera con ciertas características en la posición de la bionave ("escudo= CreateSphere (2, bionave)") y la introducimos en el mapa de colisiones como si fuera un disparo. Ya en la función de actualización ("escudo_bionave()") controlamos su aspecto ("TurnEntity") y el tiempo de activación por medio de las variables globales "Tiempo_escudo" y "Tiempo_escudo_total". Esta última variable es utilizada como acumulador de tiempo extra en caso de recoger bonus, etc. y para eliminar la cantidad de 100 unidades cada vez que el escudo es usado (Fig. 7).

(■) ACTIVACIÓN DEL CAMUFLAJE

Para realizar el camuflaje de la bionave utilizaremos el comando "EntityAlpha". Haremos que la bionave quede transparente o invisible (en la vista subjetiva).

Su actividad, regulada por la función "Camuflaje_bionave()", es controlada, al igual que con el escudo, por los contadores "Tiempo_camuflaje" y "Tiempo_camuflaje_total".

En el próximo número...

... estudiaremos el sistema de colisiones que le permite actuar con el entorno.

Fabricando los elementos del **juego** (III)

En esta entrega daremos el aspecto definitivo a la planta carnívora Dreeck que modelamos en el número anterior y le proporcionaremos movimiento con Character FX.

TEXTURIZANDO EL DREECK CON DEEP PAINT 3D

Una vez dentro del programa, cargamos nuestro modelo "Dreeck.obj", observamos que en la ventana *Material Import* se indica la presencia de un mapeado UV. Creamos un material vacío para poder continuar pinchando en el primer cuadro de la sección *Material* en *Unknown* y elegimos un tamaño de 256 x 256 y como nuevo canal de color ("C") seleccionamos en *Add New Channel*.

Para texturizar la planta carnívora vamos a utilizar algunas opciones que todavía no hemos estudiado. No son realmente necesarias para la textura final, pero ayudarán a obtener un mejor aspecto. Se trata de utilizar más canales aparte del color, como "Bump" (protuberancias) y "Shine" (brillo).

Para activar estos canales debemos ir al panel de comandos (Panel Command) y seleccionamos en *Elements* (F7) y luego en *Layers* (Capas). Aparecerá una capa base con cinco casillas (C, B, G, S y O) que representan la paleta del material. Cuando activemos las casillas "B" y "S" nos preguntará por un tono de color, elegimos uno claro. El "Bump" proporcionará profundidad a la textura y "Shine" un aspecto más viscoso (Fig. 1).

Ya estamos preparados para pintar la planta. Comenzamos por la boca. Utilizando la lupa centramos esta parte del modelo en la pantalla. Elegimos el pincel en *Presets* (F5) y para la textura la piel *Lizard Skin+* de la lista *Skin, Furs and Hair*. Seguidamente, modificamos el color y el tamaño de la piel elegida. Nos situamos en *Brush and Paint Settings* y en la pestaña *Base* de *Advanced Behavior* cambiamos la tonalidad (Hue) a verde (-215) y un tamaño en *Scale X* de 0.69 para hacer las escamas de la piel más pequeñas. Luego, más arriba, en *Brush Settings* colocamos la pluma (Feather) a 0 y una fuerza de presión (Strenght) al 100% (Fig. 2).



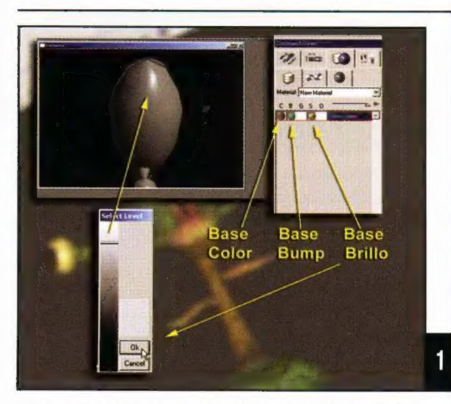
NOTA

Cada canal creado genera una textura independiente; es decir, si activamos el canal del color, brillo y bump, se crearán tres texturas que será posible unir por el método "blending". Evidentemente, mezclar varias texturas en un objeto supone un mayor gasto de memoria y rendimiento, a cambio, claro está, de un mejor aspecto.

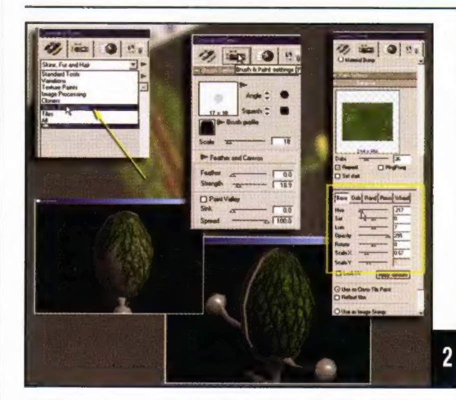


RECORDATORIO

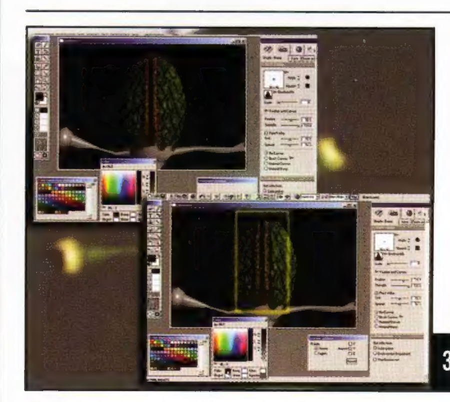
Las letras de las casillas de las capas del material son: "C" Color, "B" Bump Mapping (profundidad), "G" Glow (resplandor), "S" Shine (brillo) y "O" Opacity (opacidad).



Se pueden obtener resultados más realistas y vistosos sumando una capa de "bump" y otra de brillo (shine).



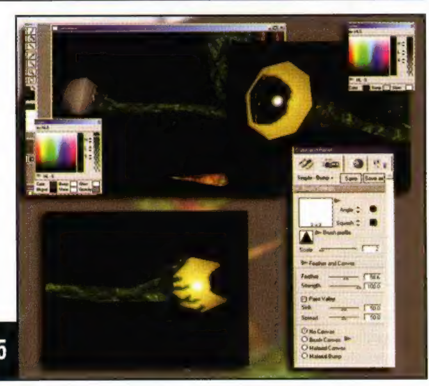
Utilizaremos un pincel y la textura de piel Lizard Skin + para pintar las mandíbulas.



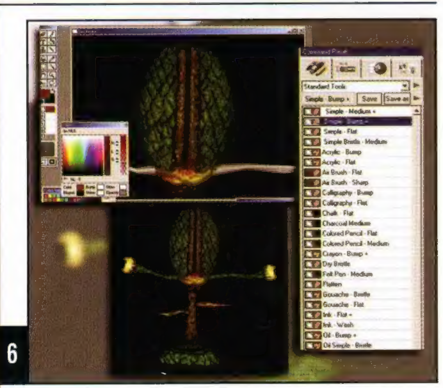
Podemos pintar los labios con un tono marrón y perfilarlos mediante una línea negra.



La herramienta de selección es muy útil para rellenar pequeñas piezas del molde como las hojas o el tallo.



Para dar mayor realismo se pueden pintar los nervios en tentáculos y globos oculares.



Unas pinceladas de tono rojizo darán un aspecto más aterrador a la planta carnívora.

Para el interior de la boca elegimos, por ejemplo, la textura *Elephant Hide* y le cambiamos el tono y la saturación para proporcionarle un color rojo a la textura. Terminamos la boca, perfilando los labios de las mandíbulas. Para ello, elegimos un color marrón en la paleta y pintamos ajustando, previamente, el pincel en tamaño y presión. Podemos aplicar más detalle al filo de las mandíbulas pintando un borde de color negro en la parte exterior del labio para proporcionar sensación de relieve (Fig. 3).

Una vez terminada la boca, pasamos a texturizar la base de la planta. Así que la seleccionamos con la herramienta de selección y rellenamos con la misma textura utilizada para la boca y con el mismo color verde. Igualmente, perfilamos los bordes de la base con el pincel con un tono de verde más oscuro y añadimos algunos trazos para simular los nervios.

El siguiente paso será pintar las hojas y el tallo. Seleccionamos una hoja y la rellenamos con la textura *FireStorm* de la lista *Texture Paints* (Fig. 4).

Seguidamente, hacemos lo mismo con la otra hoja y el tallo. Para los tentáculos oculares utilizamos la misma textura que el tronco pero con el color modificado. También en esta parte del modelo pintaremos trazos de color verde para simular los nervios. Para terminar, seleccionamos los ojos con la varita mágica de la herramienta de selección y rellenamos con un color amarillo. Luego, con el pincel pintamos de negro el iris.
Podemos añadir algunos trazos en el globo ocular, en este caso también de color verde, para simular nervios o venas (Fig. 5).

Para aumentar el detalle general, pintamos algunas manchas de sangre en la boca con el pincel *Simple bump+* (Fig. 6).

Ya hemos terminado de pintar nuestro modelo. Ahora sólo nos queda salvar la textura. Si únicamente vamos a utilizar el canal del color, podemos salvar la textura pulsando con el botón derecho del ratón sobre la letra "C" y elegir la opción del menú flotante *Export Channel*. Podemos hacer lo propio para cada canal o bien a través de la opción *Save All Maps* en el menú *File*. En ambos casos elegiremos como nombre de archivo "textura_dreeck.bmp".

TRUCO

Pulsando la tecla de tabulación "Tab" ocultaremos todas las ventanas de herramientas y podremos ver el modelo terminado previamente ajustado en la ventana con "Ctrl"+ 0 (cero).

CREANDO LA ANIMACIÓN CON CHARACTER FX

Para terminar con nuestra planta carnívora vamos a asignarle algunos movimientos con Character FX. Una vez dentro del programa, cargamos el modelo "dreeck.obj" que tenemos guardado. Si no se dispone de él, es posible encontrarlo en "Extras" del CD. Una vez cargado, lo ajustamos a la pantalla y configuramos la vista de la ventana 3D en "Left". Seguidamente, le asignaremos la textura que hemos dibujado anteriormente.
Seleccionamos todo el modelo y entramos en el editor de materiales (Material Editor) en el menú "Windows". Cargamos la textura "textura_dreeck.bmp" (se puede encontrar también en "Extras" del CD) y la asignamos al modelo. A continuación, elegimos la opción de visualización *Textured* en la ventana 3D para ver la planta texturizada (Fig. 7).

CREANDO EL ESQUELETO

Antes de crear el esqueleto debemos tener en cuenta qué movimientos tendrá la planta y qué partes de ella se moverán. Según el diseño, la planta será capaz de girar sobre sí misma, estirar y encoger el tallo y abrir o cerrar las mandíbulas. Con esta información, sabemos que debemos crear un esqueleto que recorra el tallo desde la base y se extienda por las mandíbulas. Necesitaremos, entonces, un punto de unión en la base, otro en medio del tallo y otro en la base de la boca. De este último, partirán dos uniones más hacia ambas mandíbulas. En la figura 8 se puede apreciar la disposición de estos puntos de unión.

Es importante colocar las uniones de abajo hacia arriba en orden. Empezaremos con la unión de la base, seguimos con la del tallo y base de la boca. A continuación, colocamos un punto en mitad de la mandíbula y otro al final. Para la mandíbula derecha, debemos partir de nuevo desde la unión de la base de la boca. Para lograrlo, debemos determinar como final de jerarquía el último punto colocado. Con esta finalidad, activaremos la casilla *IK Chain Terminator* en la ventana flotante de opciones *Tool Option* en la pestaña *Joint*. Seguidamente, con la herramienta de selección seleccionamos la unión de la base de la boca. A partir de ahora, ya podemos colocar las uniones en la mandíbula derecha (Fig. 8).

ASIGNANDO VÉRTICES

Una vez creado el esqueleto, es necesario asignar los vértices a cada unión. En este punto debemos hacer un inciso para explicar un consejo necesario para la buena implementación de este procedimiento en el caso de querer exportar la animación en formato .B3D para la versión 1.76 o superior de Blitz3D. Si una unión inferior (en la jerarquía) debe afectar a todas las uniones superiores, sólo es necesario asignar los vértices que NO son asignados a otras uniones.

Si no se siguiera este procedimiento, la animación dentro de Blitz3D no funcionaría. Para otro formato como .MD2 no hay ningún tipo de limitación.

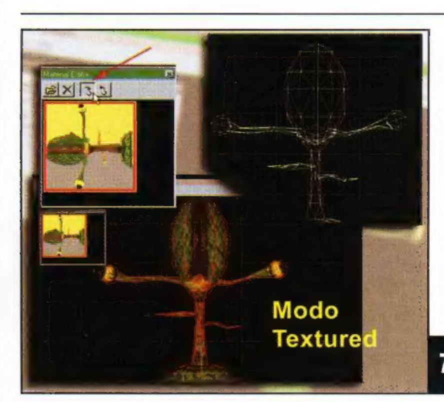
Por ejemplo, en nuestro caso, la unión del centro del tallo afectará al resto de la planta, así que lo más normal sería asignarle todos lo vértices a esta unión. Sin embargo, más arriba en la jerarquía, tenemos otras dos uniones con asignaciones, las que abren y cierran la boca. Pues bien, no es necesario incluir todos los vértices asignados a estas dos uniones. En la figura 8 se puede apreciar las asignaciones de vértices siguiendo este procedimiento. Una vez que hayamos asignado los vértices a las uniones, conmutamos al modo de animación *Animation* para comprobar los movimientos.

CREANDO LA ANIMACIÓN

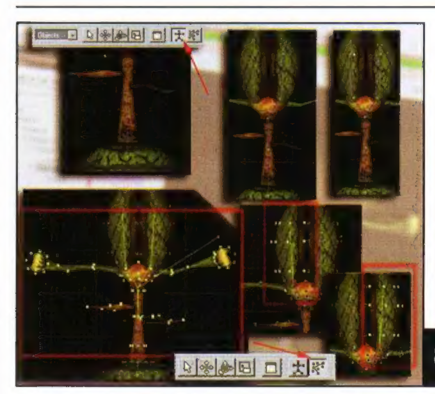
Una vez que hemos comprobado que todas las asignaciones son correctas y que el esqueleto creado cumple con nuestras expectativas de diseño, pasamos a realizar el script de animación en el *Keyframer*. Antes de empezar, vamos a situar la vista en el modo *Perspective* y ajustamos el modelo con la herramienta de zoom y desplazamiento de la ventana 3D. Para que Character FX ejecute la animación a una velocidad más o menos moderada, vamos a cambiar el *Frame Rate* a 15, es decir, 15 fotogramas por segundo. Para ello, selecciona-

RECORDATORIO

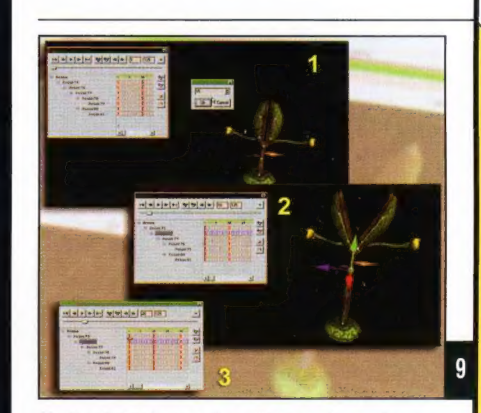
Para sumar más fotogramas a la línea de tiempo, escribimos el número de frames en la casilla de "frame final" y pulsamos "Enter".



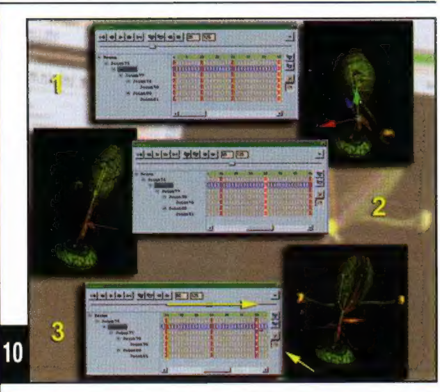
El primer paso para prepara nuestro modelo en Character FX es preparar la vista y texturizar el modelo.



Debemos seguir un orden secuencial a la hora de crear la jerarquía en el esqueleto y saber qué vértices corresponden a una unión.

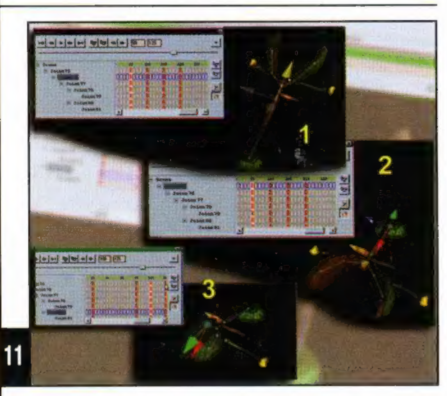


Copiando keyframes de un lugar a otro podemos lograr movimientos cíclicos perfectos.



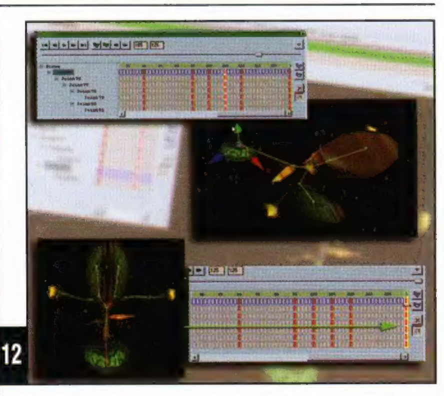
10

Seleccionando la unión del tallo giraremos toda la planta.



11

En un mismo keyframe podemos sumar movimientos de distintas uniones.



12

La velocidad de paso de un estado a otro en la animación dependerá de la cantidad de fotogramas entre keyframes.

mos la opción *Custom* en *Animation \ Set framerate* y colocamos un 15.

En la ventana *Keyframer* nos situamos en el primer fotograma y creamos un keyframe nuevo pulsando en . Seguidamente, avanzamos 10 frames en la línea de tiempo y creamos otro keyframe para albergar el primer movimiento. Éste consiste en subir y bajar la planta alargando el tallo. Para realizar este movimiento, nos situamos en el último keyframe creado y con la herramienta de traslación desplazamos la unión hacia arriba pulsando en la flecha verde (eje Y). En ese mismo fotograma y con la planta desplazada hacia arriba, vamos a abrir las mandíbulas. Para ello, seleccionamos la primera unión de la mandíbula izquierda y con la herramienta de rotación la hacemos girar pulsando en la flecha roja (eje X). Hacemos lo propio con la mandíbula derecha. Una vez que hemos abierto la boca pulsamos en el botón de la llave de *Set keyframe*. Seguimos, activando el modo de copia pulsando en el icono , seleccionamos todo el frame primero y lo movemos hacia el fotograma 20. La postura primera de reposo se copiará. De esta forma volvemos a mover la planta a su posición inicial perfectamente (Fig. 9).

El siguiente paso será crear el modo de búsqueda de comida que adopta la planta, consistente en observar hacia ambos lados girando el cuerpo de un lado a otro. Creamos un nuevo keyframe en el fotograma 30 y nos situamos en él. A continuación, con la herramienta de rotación, hacemos girar la planta desde la unión del tallo hasta la posición que se muestra en la figura 10. Para volver de nuevo a la posición original, realizamos la misma operación de copia, pasando el fotograma 20 al 50. En el 65 colocamos el keyframe del giro hacia la derecha y en el 80 situamos una copia de la postura del fotograma 50 (Fig. 10).

Nos queda completar el modo de ataque. Aquí, la planta se alargará exageradamente a la vez que gira hacia abajo para agarrar la víctima con las mandíbulas. Necesitaremos realizar varios movimientos en el mismo fotograma. Asignamos un nuevo keyframe en el fotograma 95 y nos situamos ahí. Para el movimiento, en primer lugar alargamos el tallo como hicimos anteriormente, pero en este caso al doble de distancia, desplazando la unión del tallo.

Luego, rotamos sobre el eje Z (flecha azul) de la unión de la base para que la planta baje hasta la altura del terreno (hacia su base) y para finalizar la acción, abrimos la boca utilizando el mismo método anterior con las mandíbulas. Para cerrar la boca, nos situaremos en el fotograma número 100, creamos un nuevo keyframe y la cerramos retrocediendo también un poco el tallo. Así, dará la impresión de que ha cogido alguna presa. Sumamos un nuevo keyframe en el fotograma 105 y nos situamos en él. Aquí, llevaremos la planta agachada rotando sobre el eje Y de la unión de la base y le abrimos la boca (Fig. 11).

Después hacemos lo mismo en el fotograma 110 y cerramos la boca. Para terminar, sumamos un nuevo keyframe en la posición 125 y copiamos ahí la posición del fotograma 1 para iniciar la animación y conseguir un ciclo perfecto.

Ya sólo nos queda salvar el modelo animado en la opción *Export* del menú *File* en el formato .MD2 o .B3D, aunque lo salvaremos en ambos (Fig. 12).



En el próximo número...

... seguiremos fabricando los elementos del juego modelando y texturizando algunos edificios.

Nuestro primer tema musical con Anvil Studio (I)

Comenzamos en este coleccionable con una serie dedicada a la creación de música para nuestros juegos mediante el secuenciador Anvil Studio. Un programa muy asequible, fácil de utilizar y con muchas posibilidades, que permite realizar cualquier tema musical a través del manejo de pistas MIDI y de muestras en formato .WAV. Anvil Studio también permite crear pistas rítmicas con nuestros sonidos de percusión favoritos y manejar librerías de sonidos sintetizados.

Es ideal para trabajar nuestros temas musicales en pistas MIDI incluso si no disponemos de ningún teclado maestro externo o sintetizador conectado al ordenador, ya que dispone de un compositor ("composer") para crear los temas nota a nota.

PRIMERA TOMA DE CONTACTO

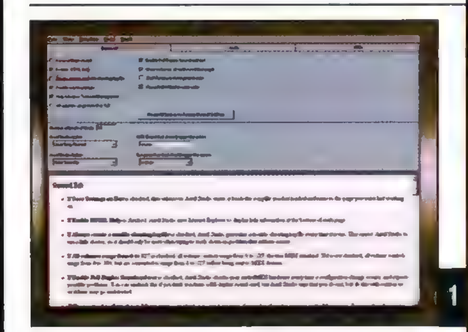
En esta entrega vamos a tener una primera toma de contacto con el programa para ir conociendo sus habilidades y facilidad de uso.

Cuando lo ejecutamos por primera vez se nos muestra interactivamente un fichero de ayuda junto a la interfaz del programa. De momento, para tener más despejada la interfaz, vamos a ocultar el fichero html de la ayuda. Elegimos la opción *View / Options*. Una vez dentro de la ventana de opciones deseleccionamos la opción *Enable HTML help* y pulsamos en *back*. Por defecto, el programa nos sitúa en la ventana *Mixer* (Fig. 1).

En esta sección es donde crearemos, editaremos y mezclaremos las distintas pistas de una canción.

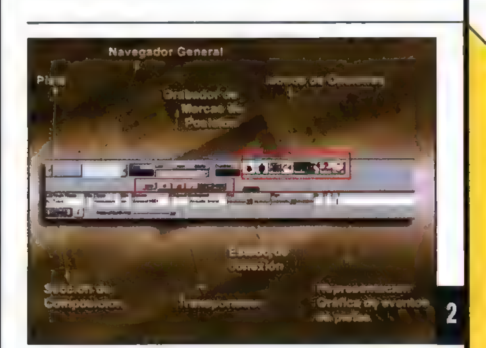
En Anvil Studio podemos trabajar con distintos tipos de pistas de manera independiente y que serán mezcladas en el mezclador (*Mixer*). Disponemos de pistas MIDI o de instrumentos, pistas de audio y pistas de ritmos. Disponemos de un transportador para reproducir y grabar y un navegador para recorrer las pistas. Disponemos también de un grabador de marcas de posición para colocarnos en diferentes puntos definidos de la canción. Asimismo, unos iconos nos permiten activar o desactivar ciertas características del programa. En la figura 3 se muestra la descripción de estos iconos.

Por defecto, sólo se muestra una pista (track), cuyos elementos pasamos a describir a continuación empezando por la izquierda. A principio de pista encontramos un cuadrado pequeño de color verde. Este color nos indica el estado de la pista que cambiará dependiendo de la casilla *On*. A continuación, en *Track Name*, podemos nombrar la pista pulsando sobre *Track 1*. Con *No* se indica el número de la pista y con *Type* a qué clase pertenece. Por defecto, aparece una pista de tipo MIDI (*instrument*), la cual podemos cambiar pulsando sobre *instrument* y eligiendo entre *Rhythm* o *Audio* (más adelante trabajaremos con cada una de ellas). En *On* definimos el estado de la pista entre *mute* (no sonará), *solo* (se mutarán las demás pistas menos la seleccionada) y *on* activa. En el apartado *Device* seleccionamos el dispositivo sonoro y dependerá del tipo de pista seleccionado. Así, por ejemplo, en una pista de tipo *instrument* aparecerá el sintetizador de sonidos correspondiente a la tarjeta en uso. Por defecto utiliza el *General MIDI*, apareciendo la lista de sonidos compatible MIDI. En el apartado *Channel* se seleccio-



1

Método para ocultar las pantallas de ayuda de la ventana de trabajo Mixer.



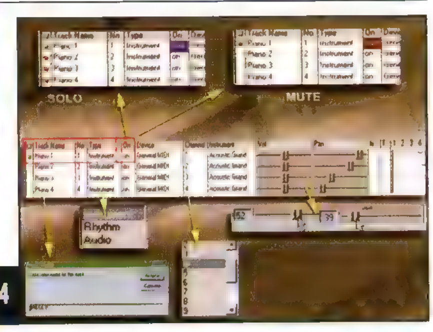
2

Descripción de los elementos del mezclador Mixer.

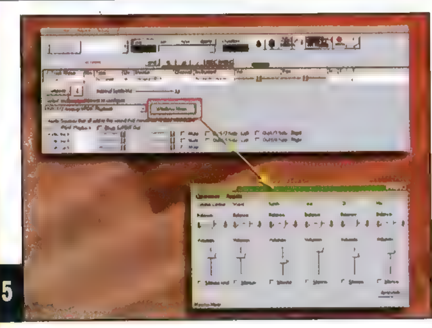


3

Descripción de los iconos de opciones del mezclador.



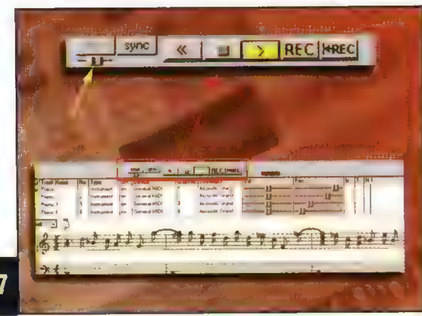
Diferentes elementos de las pistas del secuenciador.



A través del pequeño botón del "altavoz" podemos acceder a la configuración de nuestra tarjeta de sonido.



En la sección composer es donde podemos editar todos los eventos de cada pista.



Algunas opciones de navegación sólo aparecen en la sección composer.

na el canal MIDI de la pista (desde 1 a 16) y en *Instrument* el sonido. Seguidamente, encontramos un deslizador para cambiar el volumen de la pista (*Vol*) y una casilla numérica para definir el panorámico, el cual se modifica a través de un deslizador de color más claro que aparece al hacer clic sobre la casilla *Pan*. Los siguientes parámetros *fx* y *Time Shift* se utilizan para pistas de audio. El primero sirve para asignar un efecto a la pista y el segundo para provocar un retardo en la ejecución de la pista con respecto a las demás. Cada una de las casillas siguientes, representadas por números, corresponden a los compases de la canción y es una representación gráfica del contenido de la pista, la cual se editará en la sección de composición *compose* (Fig. 4).

Para modificar las opciones generales de audio; es decir, la configuración de la tarjeta y del mezclador de Windows, debemos pulsar en el icono . Pulsándolo de nuevo ocultaremos todos los parámetros. Asimismo, podemos modificar el volumen general del sintetizador de la tarjeta con el deslizador junto al icono de configuración "Internal Synth Vol" (Fig.5).

Para crear una nueva pista elegimos la opción *Create* del menú *Track* y a continuación el tipo de pista. Si lo que queremos es borrar el contenido de una pista elegimos la opción *Erase Notes* o *Erase Notes and Properties* del menú *Track*. Para borrar la pista completa elegimos *Delete*. Para hacer una copia completa y exacta de una pista utilizaremos *Clone*. Se creará una pista nueva. También podemos mezclar el contenido de dos pistas con la opción *Merge*. Al elegirla, el programa nos pedirá con qué pista queremos mezclar la seleccionada. Además, nos permitirá mezclar sólo las notas y no los demás eventos seleccionando la opción *Merge Note events only*.

Siguiendo con las opciones del menú *Track*, disponemos de una herramienta para cuantizar toda una pista, se trata de la opción *Quantitize Entire Track*. Además, con la opción *transpose* podemos transportar una sola pista a elegir (*Entire Track*) o toda la canción (*All Tracks in Song*).

PRACTICANDO CON UN EJEMPLO

Vamos a cargar una canción desde disco. Elijamos la opción *Open Song* del menú *File* y buscamos el archivo *FugueGM* que se encuentra en "Extras" del CD adjunto. Se trata de un tema clásico en formato MIDI repartido en 4 pistas. Observamos que en las casillas numeradas de las pistas aparecen una serie de puntitos, son los eventos MIDI de cada compás. Para recorrer los compases debemos utilizar el navegador (gran barra deslizadora de a la izquierda del programa). Para seleccionar una u otra pista podemos utilizar las flechas del cursor. El contenido de cada pista sólo podemos verlo a través de la sección *Compose*, la cual se activa pulsando en el botón "Compose". Seleccionemos la pista número 1 y activemos "Compose". Observamos que aparece un pentagrama con el contenido de música de la pista. Desde aquí, tenemos más opciones para navegar por la pista además del navegador principal. Podemos ir hacia la derecha o izquierda con los cursores y al principio o fin de la pista con "Ctrl" + Cursores. Podemos seleccionar con el ratón cualquier parte del pentagrama o bien con "Shift" + Cursores (Fig. 6).

Para reproducir la canción sólo tenemos que pulsar en el icono y para avanzar paso a paso en la canción disponemos de la opción "step" situado en un pequeño botón a la izquierda del transportador. El pequeño deslizador situado debajo de *step* es para aumentar o disminuir la cantidad de notas de cada paso (Fig. 7).



En el próximo número...

... empezaremos a trabajar con Anvil Studio para crear nuestra primera canción.

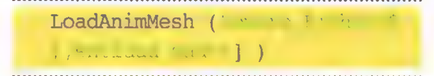
Manejo de funciones 3D (II). Animación

En la mayoría de las aplicaciones multimedia y sobre todo en los videojuegos se pueden encontrar geometrías en 3D con algún tipo de movimiento conviviendo con objetos estáticos.

Blitz3D contempla también la posibilidad de controlar modelos con alguna animación, realizada en aplicaciones especiales como CharacterFX. Pero además, también es posible crear nuestra propia animación desde programación. Estudiaremos funciones estándar para la manipulación y creación de animaciones y funciones nuevas añadidas en versiones posteriores a la 1.66.

MANIPULANDO OBJETOS CON ANIMACIÓN

El primer método depende de la secuencia de animación contenida en un modelo 3D y consiste en cargar el *mesh* y con él toda la animación e información de jerarquías (si las tuviese). Para ello, debemos utilizar el comando "LoadAnimMesh":



```
LoadAnimMesh (
[, ] )
```

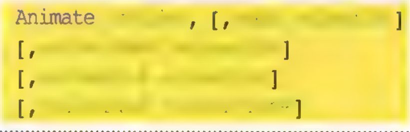
Esta instrucción cargará cualquier fichero en los formatos .X, .3DS y .B3D. tanto si tuviera animación como si no.

La animación contenida en un objeto 3D está distribuida en secuencias. Quiere esto decir que si, por ejemplo, tenemos el modelo de un ogro, el cual anda, corre y salta, cada una de estas acciones vendrán contenidas en el fichero una detrás de otra, cubriendo un número determinado de fotogramas cada una. Blitz3D al cargar este modelo obtendrá la animación completa, es decir, todos los fotogramas unidos:



```
= LoadAnimMesh ("  ")
```

Para ejecutar la animación contenida en el modelo utilizaremos la función "Animate":



```
Animate  , [,  ]
[,  ]
[,  ]
[,  ]
```

Esta función extraerá automáticamente la animación contenida en el modelo y la reproducirá continuamente, desde el primer fotograma hasta el último.

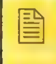
Sólo tenemos que llamarla una sola vez y actualizarla con "UpdateWorld" (Ver "ejemplo1.bb").

Con escribir: *Animate Modelo* ya funcionará. Sin embargo, posee algunos parámetros opcionales que nos proporcionan el control de la animación de una forma más flexible.



NOTA

Blitz3D puede manipular objetos animados en los formatos .X (DirectX), .3DS (3D Studio Max), .MD2 (Quake 2) y el reciente formato de Blitz3D para la versión 1.75 y superior, .B3D (muy similar al formato .MD3 y soporta animación por esqueletos, varios mapeados UV y múltiples capas de texturas). (Ver "ejemplo5.bb").



NOTA

La función "UpdateWorld" tiene un parámetro opcional para controlar la velocidad general de todas las animaciones del programa: *UpdateWorld [velocidad_animación#].*



Mediante la función "Animate" podemos reproducir la animación contenida en el modelo cargado con "LoadAnimMesh".



Podemos reproducir la animación de diferentes maneras según el valor del segundo parámetro de la función "Animate".

El parámetro *modo animación* va desde 0 a 3 y define cómo se reproducirá la animación: (Ver "ejemplo2.bb").

- **0.** Animación parada.
- **1.** Animación en ciclo. (Por defecto)
- **2.** Animación en ciclo hacia delante y atrás sucesivamente.
- **3.** Se reproduce la animación una sola vez.

La "velocidad de animación" es un número "float" que determina el "framerate" (fotogramas en un segundo) o velocidad de la animación al reproducirse (Ver "ejemplo3.bb").

El parámetro "Secuencia de animación" especifica qué secuencia de fotogramas contenida en la animación se reproducirá. Por defecto, la primera secuencia adquiere un valor de 0. Este parámetro se utiliza para encadenar animaciones. Ahora bien: inicialmente, un modelo o entidad cargado con "LoadAnimMesh" dispondrá de una sola secuencia de animación. Por lo tanto, aunque nuestro modelo de "ogro" anterior contuviera tres secuencias diferentes, no podríamos distinguirlas desde Blitz3D, ya que se consideran como una sola animación.

Para resolver este problema en versiones de Blitz3D anteriores a la 1.75 es necesario tener varios ficheros con una secuencia de animación diferente en cada uno de ellos, por ejemplo, "ogro_andando.3ds", "ogro_corriendo.3ds", etc. Luego, se cargarían uno a uno los ficheros y se encadenarían las animaciones en una misma entidad con la instrucción "LoadAnimSeq":

```
Modelo_ogro = LoadAnimMesh
("ogro_andando.3ds")
LoadAnimSeq Modelo_ogro,
"ogro_corriendo.3ds"
LoadAnimSeq Modelo_ogro,
"ogro_saltando.3ds"
If keydown(205)
  Animate Modelo_ogro,1,.5,1,30
; Ogro  corriendo (secuencia 1)
Else
  Animate Modelo_ogro,1,.5,0,30
; Ogro andando (secuencia 0)
Endif
If keydown(205)  Animate
Modelo_ogro,1,.5,2,30
; Ogro saltando (secuencia 2)
```

NOTA

En la función "Animate", un valor negativo en el parámetro de la velocidad provocaría la reproducción hacia atrás de la animación.

```
If KeyDown(2)
   Velocidad#=.5
   Animate Modelo,1,Velocidad,Secuencia,Transicion
Endif
If KeyDown(3)
   Velocidad#=1
   Animate Modelo,1,Velocidad,Secuencia,Transicion
Endif
```

3

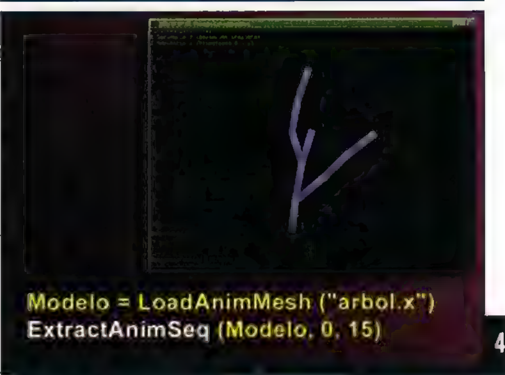
Es posible modificar la velocidad de reproducción de la animación por medio del último parámetro de la función "Animate".

En este ejemplo, hemos encadenado a la variable manipuladora "Modelo_ogro" tres animaciones diferentes provenientes de tres ficheros.

Hemos construido una animación mayor con tres secuencias, las cuales ya son posibles de diferenciar:

- **Secuencia 0** = Andando
- **Secuencia 1** = Corriendo
- **Secuencia 2** = Saltando

Así que, en la sentencia "Animate Modelo_ogro, 1, .5, 1, 30 ", animamos el modelo del ogro continuamente (primer parámetro) a una velocidad de .5, con la secuencia número 1 (corriendo) El último parámetro (30) se refiere al valor de transición entre una secuencia y otra. Si se coloca un valor 0 de transición, el paso de una secuencia a otra se haría inmediatamente. Mientras que si colocamos un valor mayor (30), se realizará el paso suavemente desde el último fotograma hasta el primero de la nueva secuencia.



La función "ExtractAnimSeq" (Versión 1.75) permite adquirir secuencias de animación de un modelo animado evitando tener que tener cada secuencia en ficheros separados.

Para evitar todo este engorroso proceso de ir sumando secuencias cargadas de ficheros, se introdujo una nueva función a partir de la versión 1.75: "ExtractAnimSeq". Este comando permite "extraer" todas las secuencias contenidas en una animación para poder ser animadas posteriormente con la instrucción "Animate":

```
ExtractAnimSeq (Entidad,
Primer fotograma, Último fotograma
[,secuencia de animación] )
```

También es posible extraer secuencias de fotogramas de una secuencia mayor.

La primera vez que usamos esta función, Blitz3D entenderá que los fotogramas extraídos constituyen la secuencia primera, es decir, la número 1 (la número 0 la constituye la animación completa cargada con "LoadAnimMesh"). La siguiente vez que se utilice, corresponderá a la secuencia segunda (2) Y así sucesivamente:

Código 1. Uso de ExtractAnimSeq

```
; Secuencia 0. Fotogramas 0 al 90. Ogro andando, corriendo y saltando
Modelo_ogro = LoadAnimMesh ( "ogro.3ds")
ExtractAnimSeq (Modelo_ogro, 0, 20)    ; Secuencia 1. Fotogramas 0 al 20.  Ogro andando
ExtractAnimSeq (Modelo_ogro, 21, 50)   ; Secuencia 2. Fotogramas 21 al 50. Ogro corriendo
ExtractAnimSeq (Modelo_ogro, 51, 90)   ; Secuencia 3. Fotogramas 51 al 90. Ogro saltando
Animate Modelo_ogro, 1, .5, 1          ; Animando continuamente al ogro andando  con velocidad .5
```

(Ver "ejemplo4.bb" o "ejemplo4.exe").

OBTENIENDO INFORMACIÓN

Para tener un control más exhaustivo de los procesos de animación, disponemos de varias instrucciones que nos ayudarán a obtener información muy valiosa.

En ocasiones, necesitaremos saber si una animación sigue su curso o ha terminado. Para saberlo, disponemos del comando "Animating", el cual retorna "True" si la animación de una entidad no ha acabado: Animating (Entidad).

```
If Animating (Modelo_ogro) =
False  Text 10,10,
"La animación ha terminado"
```

NOTA

Cuando se carga un modelo .MD2 sólo lo hace la geometría, las texturas no. Por ello, es necesario cargarlas y asignarlas a parte.

El nuevo formato ".B3D", soportado a partir de la versión V. 1.75, permite la animación por esqueletos, varios mapeados UV y múltiples capas de textura.

Podemos también saber cuántos fotogramas tiene la animación de un fichero cargado mediante la instrucción "AnimLenght". Esta función devuelve un número entero correspondiente a la longitud de la animación de una entidad:

```
Text 10,10,"El número de
fotogramas es: "+AnimLenght
(Modelo_ogro)
```

Para saber en qué fotograma se encuentra la animación en curso podemos utilizar "AnimTime". Esta función devuelve un valor flotante correspondiente al fotograma en curso:

```
AnimTime#  ( Entidad )
```

Por último, disponemos de otro comando que nos devolverá el número de la secuencia actualmente en reproducción: *AnimSeq (Entidad).*

ANIMACIÓN EN FORMATO .MD2

Además de los formatos explicados anteriormente, Blitz3D permite la manipulación de modelos con animación en formato .MD2. Los comandos normales de animación no funcionan para este tipo de modelo ya que el formato .MD2 tiene su propio grupo de comandos de animación.

Para cargar un modelo .MD2 disponemos del comando:

```
Modelo = LoadMD2 ( Fichero MD2
[, Entidad madre] )
```

Al cargar el modelo toda la animación e información de jerarquías se cargará con él. Las secuencias se almacenan de forma secuencial, así que sólo tenemos que saber en qué fotograma empieza y termina cada secuencia. Por este motivo la función para animar el modelo cambia con respecto a la función estándar "Animate". Disponemos entonces de "AnimateMD2", que anima la secuencia del modelo contenida entre fotogramas.

```
AnimateMD2 modelo MD2 [,modo
animación] [,velocidad#]
[,primer fotograma] [,último
fotograma] [,transición#]
```

Los parámetros funcionan exactamente igual que en las funciones de animación estándar.

```
Modelo_ogro = LoadMD2 ("ogro.md2")
Textura_ogro = LoadTexture
("textura_ogro.bmp")
EntityTexture Modelo_ogro,
Textura_ogro
```



Blitz3D permite el manejo de modelos en formato .MD2. Utiliza un sistema diferente de animación, por lo que las funciones para utilizarlo son distintas de las normales.

```
AnimateMD2 Modelo ogro,1,.5,(
; Ogro andando continuamente
```

Disponemos además de comandos propios para obtener información acerca de las animaciones en curso. Para obtener el momento exacto donde se encuentra la animación, disponemos de la función:

```
MD2AnimTime# (          )
```

Por ejemplo, si la animación se encuentra en la mitad del fotograma 10 y 11 obtendremos un valor de 10.5. Para saber los fotogramas totales de una animación de un modelo MD2 utilizaremos:

```
MD2AnimLength (          )
```

Y por último, para saber si una animación ha terminado o no:

```
MD2Animating (          )
```

(Ver "ejemplo6.bb").

CREANDO NUESTRAS PROPIAS ANIMACIONES DESDE PROGRAMACIÓN

También es posible generar animaciones con primitivas u otros objetos 3D con el uso de la programación. Vamos a trabajar como lo hace una aplicación para generar animaciones como, por ejemplo, CharacterFX, utilizando "Keyframes". El procedimiento es bien sencillo.

Supongamos que tenemos un cubo en una posición de la pantalla y queremos crear una animación, consistente en 30 fotogramas, en los que el cubo se desplaza hacia otra posición. Pues bien, situamos el cubo en posición:

```
 = CreateCube()
PositionEntity
SetAnimKey
```

Luego creamos un keyframe en la posición original (0) con "SetAnimKey".

Desplazamos el cubo y grabamos otro keyframe en el fotograma 30 en la nueva posición:

```
PositionEntity
SetAnimKey
```

Blitz3D creará los fotogramas intermedios automáticamente basándose en los valores de posición, rotación y traslación de los objetos en los puntos definidos por los "keyframes".

Una vez creados los puntos de la animación, la creamos con *AddAnimSeq Entidad, longitud en fotogramas:*

```
AddAnimSeq
```

Cada vez que apliquemos esta función se creará una secuencia nueva, siempre en orden secuencial. Por lo tanto, si después de desplazar el cubo queremos rotarlo y desplazarlo a la vez hacia otro lugar, tendremos que crear otro keyframe y sumar la animación:

```
PositionEntity
RotateEntity
SetAnimKey
AddAnimSeq
```

En este caso tendremos dos secuencias de animación, una desde el fotograma 0 al 30 (secuencia número 0) y otra desde el 31 al 60 (secuencia número 1). Para animar, sólo tenemos que utilizar la función "Animate" como se ha descrito anteriormente.

A continuación se muestran con detalle todos los parámetros de "SetAnimKey":

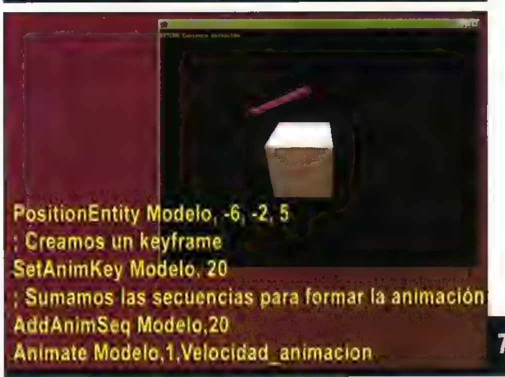
```
SetAnimKey
[,                                   ]
[,                                  ]
[,                                 ]
```

Los parámetros opcionales de activación están por defecto a 1 ("True") y sirven para incluir en el keyframe los cambios de posición, rotación y tamaño del objeto. Si en el ejemplo anterior, en el segundo "Keyframes" pusiéramos:

```
PositionEntity
SetAnimKey                False,
True, True
```

el cubo no se desplazaría en absoluto hacia la nueva posición.

Esta manera de crear animaciones es muy interesante sobre todo para poder construir nuestras propias aplicaciones de grabación de movimiento en Blitz3D. (Ver "ejemplo7.bb").



La creación de animación en tiempo de ejecución por sistema de keyframe también es posible en Blitz3D.

NOTA

Para crear una animación desde programación es necesario seguir los siguientes pasos:
1. Posición inicial modelo + "SetAnimKey modelo,fotogramas-A". Primera secuencia.
2. Siguiente posición modelo + "SetAnimKey modelo, siguientes fotogramas-B". Segunda secuencia.
3. Sumar secuencias para crear animación con "AddAnimSeq modelo, fotogramas-A+fotogramas-B.



En el próximo número...

... estudiaremos las funciones para el manejo de cámaras.

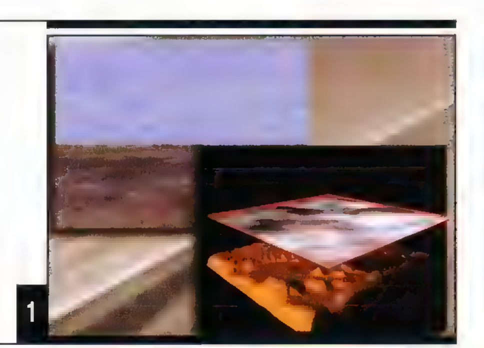
Creación de cielos para un entorno

Hay muchos métodos para la creación de cielos para un entorno. Probablemente, el más fácil sea mediante el uso de un plano situado a una altura determinada del terreno. Este método ha sido utilizado en nuestro juego "Zone of Fighters".

Otro método sería utilizar una esfera o un cubo con textura que cubra todo el entorno. Y por último, el método más sofisticado y quizás el más utilizado hoy día en videojuegos sea mediante el mapeado cúbico.

UTILIZANDO PLANOS

Utilizar un plano texturizado es un método muy sencillo, eficaz y fácil de implementar en un juego. Generalmente, se suele usar en niveles de interior en donde sólo se ve el cielo a través de las ventanas o huecos en el techo. También se puede utilizar en entornos de exterior en donde se impide ver el horizonte por el uso de niebla. Además, es muy común el uso de scroll de textura para generar la sensación de movimiento de nubes. En la figura 1 se muestra un esquema gráfico de cómo se utiliza (Ver ejemplo "cielo_plano.bb").



1

Esquema y resultado de un cielo creado por el sistema del plano.

El procedimiento básico es crear un *mesh* plano, texturizarlo y situarlo sobre el terreno o nivel (BSP, mesh, etc.):

```
Cielo=CreatePlane()
TextureEntity Cielo,Textura_cielo
FlipMesh Cielo
PositionEntity Cielo,0,1000,0
```

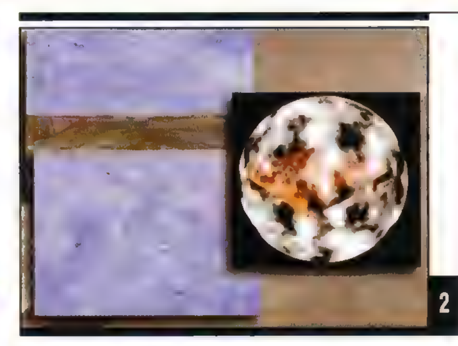
En este caso, "FlipMesh Cielo" funciona igual que "RotateEntity Cielo,180,0,0". Lo que hacemos es dar la vuelta al plano para que la textura mire hacia abajo.

Seguidamente, podemos desplazar la textura para simular el movimiento del cielo.

UTILIZANDO ESFERAS O CUBOS

La utilización de una esfera produce un resultado realmente interesante, ya que podemos mover el cielo girando el *mesh* o desplazando la textura. En el uso de esferas hay que tener en cuenta los vértices de unión de los segmentos, sobre todo por la deformación que la textura sufre en los polos de la esfera. Básicamente, consiste en posicionar el terreno en el interior de una gran esfera texturizada. Al estar en el interior no veremos la textura a no ser que le apliquemos la función "FlipMesh" a la primitiva (Ver ejemplo "cielo_esfera.bb"):

```
Cielo=CreateSphere()
ScaleMesh Cielo,
escala_X,escala_Y,escala_Z
```



2

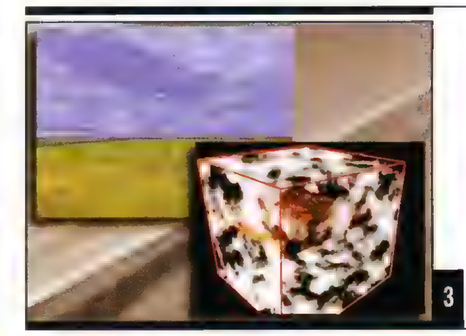
Esquema y resultado de un cielo creado por el sistema de esfera.

```
TextureEntity Cielo,Textura_cielo
FlipMesh Cielo
PositionEntity Cielo,0,1000,0
```

Por otro lado, la utilización de un cubo funciona exactamente igual, a diferencia, claro está, del resultado visual. (Ver ejemplo "cielo_cubo.bb").

MAPEADO CÚBICO

Este método consiste en crear un cubo polígono a polígono y texturizar cada una de las caras con una textura diferente. Es el más eficaz y utilizado, ya que el uso de primitivas como esferas o cubos tiene un par de inconvenientes. En primer lugar, no podemos utilizar varias texturas



3

Esquema y resultado de un cielo creado por el sistema del cubo.



Textura desglosada para un mapeado cúbico.

para generar un cielo completo; es decir, mostrando el sol y nubes distintas según se mire. Si utilizamos una esfera, la textura se adaptará a ella cubriéndola por completo y si usamos un cubo, se aplicará automáticamente en cada una de las caras. Por lo tanto, no tenemos ningún tipo de control al respecto.

En segundo lugar, al utilizar "FlipMesh" en un cubo texturizado las uniones de las caras no quedan perfectas y es posible apreciar, cuando la cámara se acerca, las aristas de color negro. Así que el mejor método resulta ser la aplicación de un mapeado por caras a un cubo modelado desde programación. Es el más complejo de todos pero los resultados son realmente buenos.



Resultado del sistema de mapeado cúbico para implementar un cielo.

PREPARANDO LAS TEXTURAS

Para este tipo de mapeado debemos preparar cinco texturas diferentes, una por cada vista del cielo: lateral izquierdo, lateral derecho, vista frontal, vista trasera y vista superior. Evidentemente, para la vista inferior no es necesaria ninguna textura, ya que nunca se vería. En la figura 4 se muestra un esquema gráfico de cómo va colocada cada una de ellas.

CREANDO Y TEXTURIZANDO EL CIELO

El siguiente proceso será modelar cada una de las caras del cubo por medio de triángulos. Éstos, a su vez, serán construidos a partir de la unión de vértices. En definitiva, crearemos una superficie por cara, con dos triángulos y cuatro vértices, y todas las superficies formarán un *mesh*. Para llevar a cabo el texturizado, es necesario la utilización de *brushes* para pintar cada superficie:

```
Cielo=CreateMesh()
;CARA FRONTAL
Textura=LoadBrush(“Textura_Frontal.png”)
Lado=CreateSurface(Cielo, Textura)
AddVertex Lado,-1,1,-1,0,0
AddVertex Lado,1,1,-1,1,0
AddVertex Lado,1,-1,-1,1,1
AddVertex Lado,-1,-1,-1,0,1
AddTriangle Lado,0,1,2
AddTriangle Lado,0,2,3
FreeBrush Textura
;CARA TRASERA
Textura=LoadBrush(“Textura_trasera.png”)
Lado=CreateSurface( Cielo,Textura)
AddVertex Lado,1,1,1,0,0
AddVertex Lado,-1,1,1,1,0
AddVertex Lado,-1,-1,1,1,1
AddVertex Lado,1,-1,1,0,1
AddTriangle Lado,0,1,2
AddTriangle Lado,0,2,3
FreeBrush Textura
...
...
```

El resto de las caras se creará de la misma manera. El procedimiento por cada lado es el siguiente:

- **1.** Se crea una superficie, la cual pintamos con el *brush* "Textura".
- **2.** Se crean cuatro vértices en esa superficie.
- **3.** Se unen los vértices de tres en tres para formar triángulos.
- **4.** Se libera de la memoria el *brush* cargado.

Para comprender mejor este procedimiento es preciso observar el esquema de la figura 5.

- **5.** Una vez creado el *mesh*, lo escalamos, posicionamos y damos la vuelta.

```
ScaleMesh Cielo,5000,5000,5000
PositionMesh Cielo,0,0,0
FlipMesh Cielo
EntityFX Cielo,1
Return Cielo
```

En el ejemplo contenido en el CD "mapeado_cubico.bb" se utiliza este sistema para rodear un terreno con un cielo y la técnica del plano para crear nubes en movimiento.

En el próximo número...

... aprenderemos a utilizar imágenes predefinidas para crear nuestra propia fuente de letras.

Juegos de estrategia (III). Estrategia en **tiempo real**

Continuando con la serie de juegos de este género tan prolífico, dedicaremos este número a aquellos títulos que no siguieron una tendencia belicista o destructiva para desarrollar su argumento, sino que tomaron como base la construcción de ciudades, fábricas, empresas o cualquier gestión de recursos a gran escala.

Pero antes, no queremos olvidar un subgénero bélico que ha levantado pasiones en multitud de seguidores de la estrategia, nos referimos a los RPS.

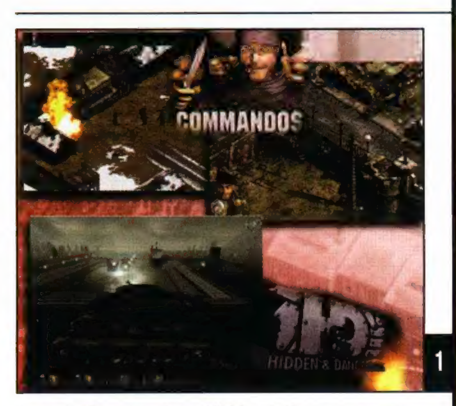
ROLE PLAYER STRATEGY (RPS)

Nacido de varios conceptos de estrategia, este tipo de juego contribuye a proporcionar a la historia de los videojuegos grandes títulos, nos referimos a los RPS. Básicamente consiste en que el jugador maneja sólo un puñado de unidades, cada una capaz de realizar una tarea determinada. Quizás el título más conocido a nivel mundial sea *Comandos: Behind Enemy Lines* (1998) y su segunda y tercera *parte Commandos 2: Men of Courage* (2001) y *Commandos 3* de la desarrolladora española Pyro Studios. La saga constituye el plato fuerte de Pyro en este género y destaca, sin duda, por la exquisita calidad gráfica de los personajes y escenarios detallados al milímetro. Este concepto de tanto éxito fue adquirido rápidamente por muchas desarrolladoras y empezaron a aparecer títulos de mayor o menor calidad gráfica pero que simulaban el mismo tipo de estrategia. Por ejemplo, en *Veiled Threat* (Lupine Games, 2000) manejamos un grupo de mercenarios en tiempos de la Segunda Guerra Mundial. También en el mismo ambiente, pero en un entorno 3D encontramos *Hidden & Dangerous* (Take2 Interactive, 2000). Sin embargo, hay dos títulos que calcan el estilo gráfico impuesto por *Comandos*, nos referimos a *Desperados: Wanted Dead or Alive* (Infogrames, 2001) o *Robin Hood: The Legend of Sherwood* (Strategy First, 2002).

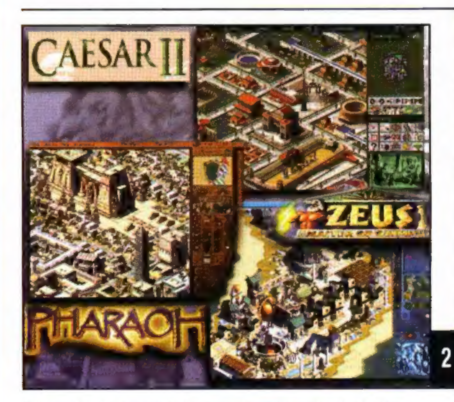
ESTRATEGIA CONSTRUCTIVA

Al igual que ocurrió en la estrategia por turnos, aparecieron también tendencias de género estratégico no belicista en el ámbito de la creación y gestión de todo tipo de ciudades a lo largo de todas las épocas. Este argumento casi no ha cambiado desde el primer constructor en tiempo real. Los conceptos se mantienen a diferencia, evidentemente, de mejoras gráficas. Debemos remontarnos al año 1989 cuando la casa Maxis saca al mercado *Sim City*. En este juego debemos construir y gestionar una ciudad entera. Un argumento aparentemente simple pero de una complejidad incuestionable que generó una gran avalancha de títulos. Maxis siguió publicando nuevas versiones de *Sim City: Sim City 2000* (1993), *Sim City 3000* (1998) y el actual *Sim City 4* (2002)

A su lado se unen otras desarrolladoras que optan por el mismo argumento como la serie *Caesar* (Sierra, *Caesar I* 1994, *Caesar II* 1995 y *Caesar III* 1998) inspirada en el imperio romano o *Pharaoh* (Sierra, 1999) en el antiguo Egipto. También apareció *Zeus: Master of Olympus* (Sierra, 2001) situada en la antigua Grecia.



La serie Commandos y Hidden & Dangerous, primeros RPS.



Ejemplo de estrategia constructiva de ciudades ambientadas en distinta épocas de la Historia.



La simulación social en The Sims adquiere matices de estrategia cuando tienes que controlar todo un vecindario.

LA BIOGRAFÍA...

WILL WRIGHT

Creador del **Shooter 3D**

Junto con Adrian Carmack y John Romero fundó ID Software, compañía creadora del shooter en 3D. Está considerado como el programador de juegos y tecnología 3D en tiempo real más importante e innovador del mundo. Su filosofía es ir siempre de la mano con los nuevos avances tecnológicos en gráficos para ordenador. Gracias a esto se ha convertido en un gurú de los videojuegos con creaciones como la saga *Wolfenstein, Doom* y *Quake*. Es amante de los viejos juegos *arcade* como *Pacman* o *Space Invaders* y aprendió solo a programar. Vendió su primer juego cuando estaba en el instituto, un RPG llamado *Shadowforge*. En 1990 entró a trabajar en SoftDisk, donde programó la serie *Commander Keen*. Su tecnología ha sido muy solicitada y el motor del *Quake II* sirvió para desarrollar juegos como *Soldier of Fortune* o *Half Life*. Carmack es amante de la enseñanza y son conocidas sus andaduras por el mundo dando infinidad de conferencias. Una muestra de esta afición es la posibilidad de adquirir por Internet el código de sus juegos más importantes.



Will Wright

La idea de crear y gestionar tu propia ciudad se extendió a otros ámbitos siguiendo la misma temática. Aparecen entonces diferentes juegos que adaptan este concepto a otras situaciones como la gestión completa de un parque de atracciones en *Theme Park* (Bullfrog, 1994) o la gestión ferroviaria que Microprose ofrece en *Railroad Tycoon* (1995), *Roller Coaster Tycoon* (1998), o la segunda parte *Railroad Tycoon 2* (Pop Top Ware, 1998) pero esta vez con gráficos en 3D. Sin embargo, aquí no queda todo. Numerosos títulos salen a la luz con multitud y disparatados argumentos. Comienza de nuevo Maxis con constructor de hormigueros *Sim Ant* (1995) y continúa de nuevo Bullfrog con su juego *Theme Hospital* (1997) donde tu misión será gestionar lo mejor posible todas las partes de un hospital. Ese mismo año Interplay publica *Evolution (*Crossover Technologies*)* en el que tienes que gestionar la evolución de las especies hasta la época de los dinosaurios. Cambiando de tercio, la desarrolladora Mucky Foot nos traslada al espacio con su trabajo *Startopia* (2001) que con unos gráficos divertidos y originales nos invita a gestionar una estación espacial repleta de extraterrestres. La buena acogida de este género, en parte debido a que son programas que pueden ser jugados por toda la familia, provoca una gran producción que conlleva a una mayor diversidad de temas. Es el caso de *Pizza Syndicate* (Software 2000, 1999), una de las más simpáticas alternativas, en el que debemos construir un imperio comercial en torno al negocio de las pizzerías. En esa misma tendencia en el sector económico, Pop Top Ware publica *Tropico* (2001), en el que tienes que controlar toda una isla.

Todo este ir y venir de juegos de estrategia económica y social provoca un seguimiento por parte de expertos de diferentes géneros interesados en utilizar este tipo de entretenimiento como motor de enseñanza en niveles profesionales reales. Es el caso del simulador de bolsa *Wall Street Trader* (Monte Cristo / Electronic Arts, 2001), considerado el mejor en su categoría.

Pero quizás el mayor fenómeno social acaecido desde la invención de los juegos de estrategia de carácter no belicista haya sido el juego *The Sims* (Maxis, 2000). En él, debemos crear y controlar a una familia. Además, podemos construir todo un vecindario y organizar todas las relaciones entre sus habitantes. Está considerado el mejor juego de tipo social y posee una IA increíble en la que se barajan casi todas las premisas que existen en las relaciones humanas. Desde su publicación, sigue estando en los primeros puestos de las listas de ventas y su mayor éxito radica en su sencilla interfaz de uso y la posibilidad de juego para toda la familia sin excepción. Actualmente, existe más de media docena de expansiones todas orientadas a ofrecer más de lo mismo y a situar a nuestras familias virtuales en diferentes lugares y situaciones: *Sim House Party, Vacation, Sims más vivos que nunca*, etc. Gráficamente todos están cortados con la misma tijera, utilizan una representación en 2D con perspectiva isométrica y con una estética bastante estilizada.

Todavía nos queda el estudio de un subgénero de estrategia donde el principal argumento es otorgar al jugador el poder de modificar la trayectoria del juego a voluntad. Se trata de los simuladores de "Dios". Sin embargo, dejaremos su estudio para el próximo número en el que también hablaremos sobre el tipo de estrategia más conocida por todos, los puzzles.



En el próximo número...

... terminaremos nuestro repaso a la historia de los juegos de estrategia en tiempo real.

Cuestionario Videojuegos



Preguntas

1. En Blitz3D, ¿para qué tipo de operaciones se utiliza la instrucción "LoadAnimMesh"?

2. Hemos creado el modelo de un hombre, el cual posee el movimiento de andar. Escribe el código para observar esa animación en Blitz3D.

3. Define una estructura simple y una función para crear disparos sólo para controlar su desplazamiento.

4. ¿Cómo podemos actualizar todos los disparos del juego en el bucle de control del juego?

5. ¿Cómo podemos determinar un punto de unión como final de jerarquía en Character FX?

6. ¿Cómo podemos variar la velocidad de reproducción de la animación en Character FX?

7. ¿Qué tipo de pistas podemos crear en Anvil Studio?

8. ¿En qué sección de Anvil Studio editamos los eventos MIDI de una pista?

9. Enumera al menos tres métodos para crear un cielo en Blitz3D.

10. Enumera las fases para programar un cielo utilizando mapeado cúbico.

Respuestas al cuestionario 11

▷ **1.** Para realizar esta operación es necesario activar el búfer de texturas con "SetBuffer TextureBuffer()" para que al dibujar los haga directamente sobre el objeto.

▷ **2.** Los *brushes* o pinceles son un grupo de propiedades del material que se le aplican a una entidad. Para crear un *brush* nuevo se utiliza la instrucción "CreateBrush" y para crearlo a partir de un fichero de imagen "LoadBrush".

▷ **3.** Para lograr una aceleración se puede utilizar una variable de desplazamiento y otra de aceleración. Además, es necesario controlar los topes máximos y mínimos en la velocidad:

```
If MouseDown(boton2)  And velocidad<velocidad_max
     velocidad# = velocidad + aceleracion#
     MoveEntity bionave, 0, 0, velocidad
  Else
     Velocidad = velocidad - (aceleracion#*2)
     MoveEntity bionave, 0, 0, velocidad
EndIf
If velocidad<4 velocidad=4
```

▷ **4.** Mediante la instrucción "PointEntity": *PointEntity camara,objetivo.*

▷ **5.** Modificando la opción *Hue* situada en la sección *Base* en la pestaña *Advanced Behaviour.*

▷ **6.** Aplicando la opción *Reverse Vertex Order* del menú *Face.*

▷ **7.** Disponemos de dos tipos de canales: canales de muestras "sampler" y de sonidos sintetizados "TS404".

▷ **8.** Primero, seleccionamos la pista o canal. Abrimos la ventana *Song settings* pulsando en *FX1* de la ventana *Channel settings.* Seleccionamos un efecto de la lista *Favorites* y pulsamos en el icono del enchufe para activar el efecto.

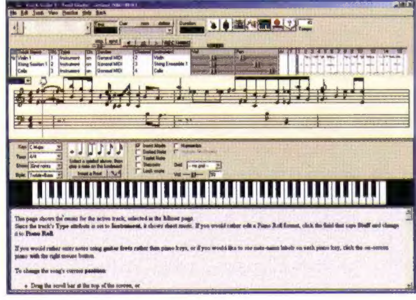
▷ **9.** Mediante la instrucción "BSPAmbientLight".

▷ **10.**
```
Linterna=CreateLight(3,camara)
LightConeAngles Linterna,0,60
```

Contenido CD-ROM 12

▶ AUDIO

■ Anvil Studio
Nueva oportunidad para hacernos con este estupendo programa con el que podrás componer tus propias melodías.



■ FractMus 2000
Con esta curiosa aplicación podrás crear música a partir de fórmulas matemáticas.

■ Mellosoftron 3.4
Convierte tu ordenador en un editor de música en vivo y crea tus propios samples.

■ Orion Virtual Studio Pro 2.3
Con este programa podremos realizar remixes como en un estudio profesional.

▶ DISEÑO 2D

■ DrawIt 3.1
Práctica herramienta de dibujo vectorial y de ilustración muy sencilla de usar.

■ Firegraphic XP 5.5.4
Edita imágenes e imprímelas usando filtros y muchas otras características.



■ Nature Painter Digital Canvas 1.1
Aprende distintas técnicas pictóricas con la ayuda de este programa.

■ PhotoWizard
Excelente herramienta para añadir sorprendentes efectos a las fotografías en muy poco tiempo.

■ AI Picture Utility 6.3
Visor de imágenes que contiene además múltiples utilidades.



■ PicViewer 2.7.4
Otro visor de imágenes en muchos formatos.

▶ DISEÑO 3D

■ AC3D 3.5
Modela y crea escenas en tres dimensiones con este programa.

■ Image Styles 3.4
Complemento ideal para la creación de los paisajes del juego en muy poco tiempo.

■ Mimmix FE 2.0
Crea simpáticos personajes en tres dimensiones con esta divertida aplicación.



■ Protix 3D Builder 1.3
Creación de imágenes en 3D con posibilidad de grabarlas en formato DirectX.

■ Character FX
Nueva oportunidad de conseguir esta excelente aplicación.

▶ PROGRAMACIÓN

■ Defect Tracker 2.0
Organiza el desarrollo del juego con esta utilidad que te permitirá reportar errores.

■ Fast Bug Track 3.1
Software de mantenimiento de cambios en los proyectos de programación.

■ Gran PM 1.6.5
Control de los proyectos y de sus errores.

■ PlayerPro SDK 5.9.4
Kit de desarrollo para poder incorporar sonido en los programas en C y C++.

▶ JUEGOS

■ Commandos: Behind Enemy Lines
Popular juego de estrategia con el que pasarás muy buenos momentos.

■ SimCity 3000
Demo del excelente juego que aún bate récords de ventas en el que deberás construir y manejar una ciudad entera.

■ Startopia
Original juego que nos invita a gestionar una estación espacial repleta de extraterrestres.



■ Zone of fighters
Nuestro juego, incluido como siempre.

▶ VÍDEO

■ Elektronika Live 1.0
Excelente aplicación para crear auténticos shows visuales en directo.



■ JPGAvi 1.0
Crea estupendos vídeos en formato AVI a partir de imágenes en JPG.

■ Video Edit Magic 1.5
Captura vídeo y edítalo cómodamente desde tu ordenador.

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.